April 10, 2015

Intelligent Agent for Open Face Chinese Poker Web-Application

Submitted May 2015 in partial fulfillment of the conditions of the award of the degree BSc (Honours) Computer Science

Alastair Kerr

axk02u

School of Computer Science and Information Technology

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature ________________

Date ____/____/____

Contents

1 Introduction **4**
1.1 Introduction to Open Face Chinese Poker . . . 4
1.2 Problem Description . . . 5
1.3 Aims and Objectives . . . 5
1.4 Ethics . . . 6

2 Background **6**
2.1 Game Theory . . . 6
2.2 Artificial Intelligence in Poker . . . 7
2.3 Hand Evaluation Algorithms . . . 7

3 Design and Implementation **8**
3.1 Approach . . . 8
3.2 Requirements . . . 8
3.3 Technologies . . . 8
3.4 Implementation . . . 8

4 Evaluation **8**
4.1 Unit Tests . . . 8
4.2 Performance of AI versus human players . . . 10
4.3 Performance of AI versus other AI . . . 10

5 Conclusion **10**
5.1 Aims and Objectives . . . 10
5.2 Reflection . . . 10
5.3 Improvements . . . 10

Abstract

Problem description: poor AI performance in OFCP, not much work done on the subject. Traditional board games and more popular variants of poker such as Texas Hold'Em and Omaha have had much more research. OFCP AIs generally underperform, playing sub-optimally. This dissertation considers different approaches and algorithms, and implements an AI for a bespoke OFC web-app.

1 Introduction

1.1 Introduction to Open Face Chinese Poker

Open Face Chinese Poker is a variant of Chinese poker where players take turns placing cards face-up into three distinct rows, creating the best possible poker hand combinations they can in order to score points from each other. Stronger hands score 'royalties' for extra points and better hands in higher rows score more points than an equivalent hand in a lower row. For example, a Royal House gives +25 bonus points in bottom row, or +50 in the middle row. Players score +1 point for each row they win in addition to any royalties. In the case that a player wins all 3 rows they score a 'scoop' bonus which grants an additional point for each row, for a total of +6 base points before royalties are calculated.

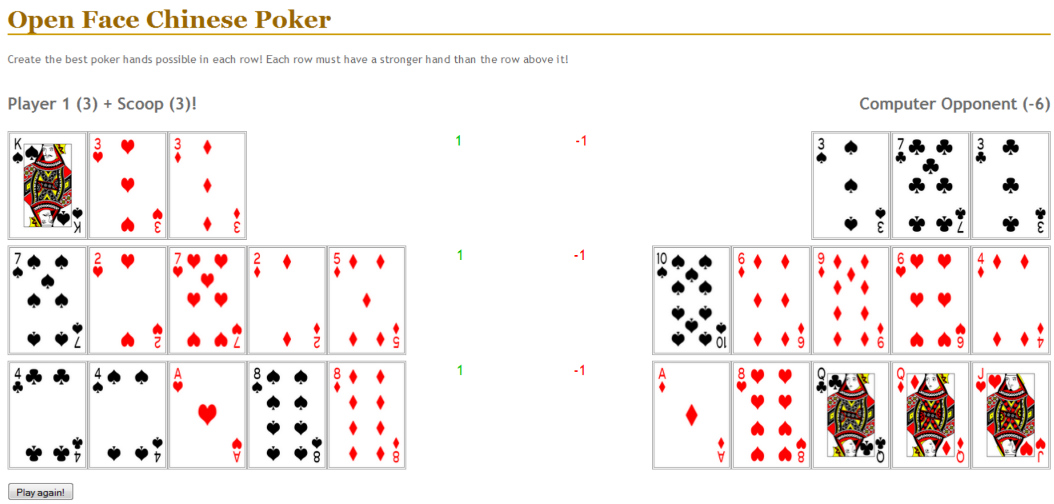


Fig 1.1.1 – example layout of the board after a single round of Open Face Chinese Poker

In this game, neither player scores any royalties because none of the hands are strong enough. Player 1 wins bottom with Two Pair 8s and 4s versus Pair of Qs for +1 point. Player 1 also wins middle row with Two Pair 7s and 2s versus Pair of 6s for +1 point. In top row Player 1 and the computer opponent both have a Pair of 3s, and so the third card is taken into account as the 'kicker', with Player 1 clinching the win for +1 point with a King kicker versus a 7. Further to the individual row scores, because Player 1 won every row they score an additional scoop bonus of +3. If a player creates a top-heavy board (i.e. a row's hand is stronger than the row below it) the player's hand is invalid and they have 'fouled'; when a player fouls their opponent automatically scoops for +6 (+1 for each row and +3 scoop bonus) in addition to any royalties.

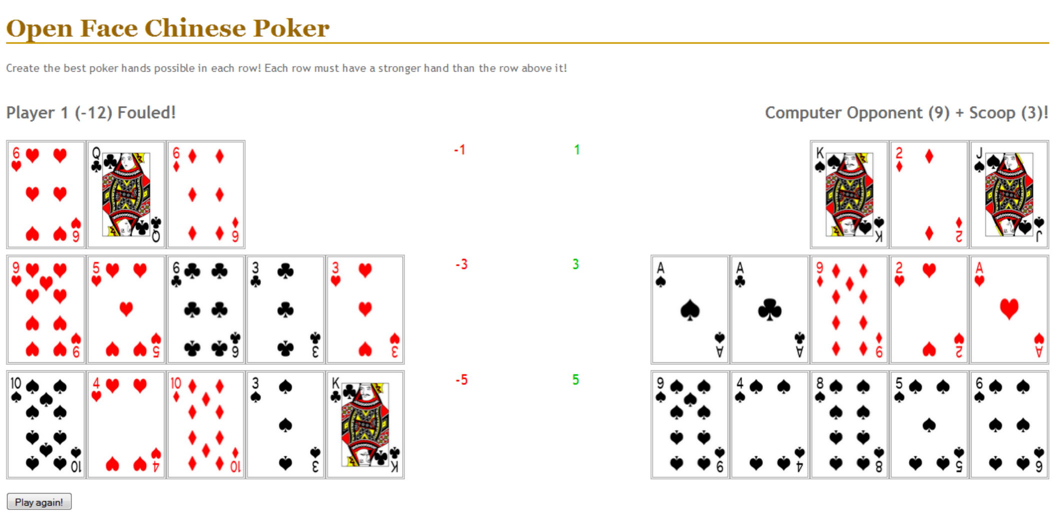


Fig 1.1.2 – Player 1 fouls and so their opponent wins an automatic scoop bonus plus royalties

Player 1 has Pair of 10s in bottom, Pair of 3s in middle and Pair of 6s in top. Because the top row contains a stronger poker hand than the row below it, the hand is invalid and the player fouls.

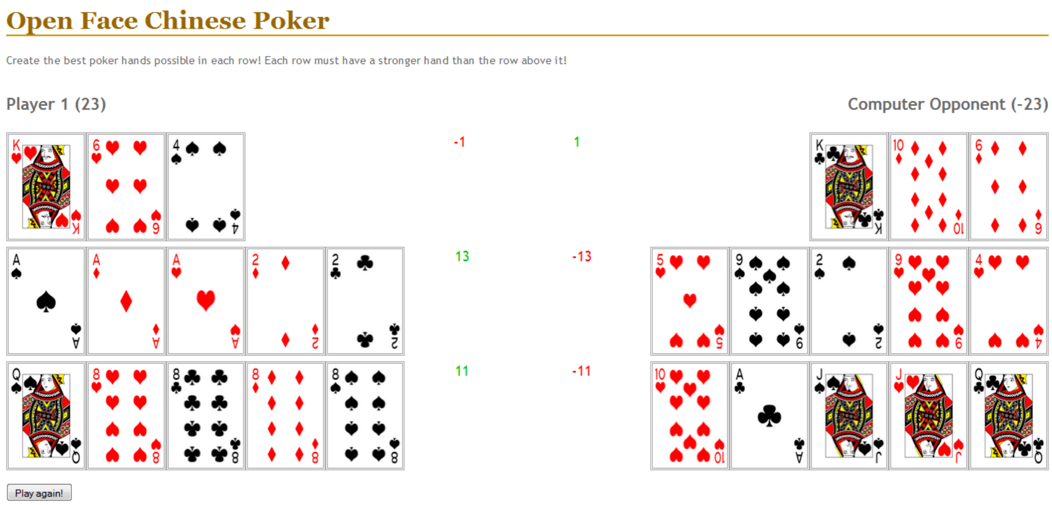


Fig 1.1.3 – Player 1 wins a lot of bonus points from royalties for strong hands

Player 1 wins bottom row with Four of a Kind 8s versus the computer opponent's Pair of Jacks. Player 1 receives +1 point for winning the row, with an additional +10 points royalty for Four of a Kind on bottom. Player 1 also wins middle row with a Full House, Aces full of Deuces versus the computer opponent's Pair of 9s, scoring +1 point for winning the row and an additional +12 royalty. Player 1 and Player 2 both have King High in the top row, but the computer opponent wins the row because their next highest card (kicker) is 10, whereas Player 1's kicker is a lower rank – 6. Player 2 wins +1 point for winning but the hand is not strong enough to score any additional royalties. The points each player wins are taken from their opponent, so the final score for Player 1 is '-1 + 13 + 11 = 23', and the computer opponent's score is the inverse of this, -23.

1.2 Problem Description

Introduce problem and motivation for study, outline purpose OFCP poker bots generally underperform vs human players. Rely on simple algorithms and make sub-optimal plays. Little work done on OFCP AI, more focus on other games/ more well-known variants of poker (Texas Hold'em). Poker is an interesting area of research for game theory as has many aspects other games (e.g. chess) do not – chance from unknown cards.

Imperfect information means that more sophisticated algorithms are generally necessary for optimal play. Or can traditional algorithms with suitable heuristics perform as well or better?

Hidden information poses many challenges from an AI point of view. In many games, the number of states within an information set can be large: for example, there are 52! $\tilde{}=$ 8x10^{67} possible orderings of a standard deck of cards, each of which may have a corresponding state in the initial information set of a card game. If states are represented in a game tree, this leads to a combinatorial explosion in branching factor.

1.3 Aims and Objectives

Specific objectives

1. Create a functional Open Face Chinese Poker Application – game environment
2. Create Intelligent Agent for application
3. AI must have a level of sophistication such that it performs well vs. humans and other AI
4. AI algorithms must be suitably optimised with tradeoffs between efficiency, space and time complexity so that it is responsive. E.g. AI calculates move in less than 5 seconds

1.4 Ethics

Gambling, bot vs humans playing for money? Or just playing for fun

2 Background

2.1 Game Theory

Game theory – John von Neumann. Traditional Game Search: e.g. 1928 neumann proposes minimax tree search. Minimax decision rules dictate that in a 2-player zero-sum perfect information game there exists strategies for each player that minimise his maximum losses (hence minimax) which must be based on considerations of all the adversary's possible responses. The strategy which minimises a player's maximum losses is called the optimal strategy.

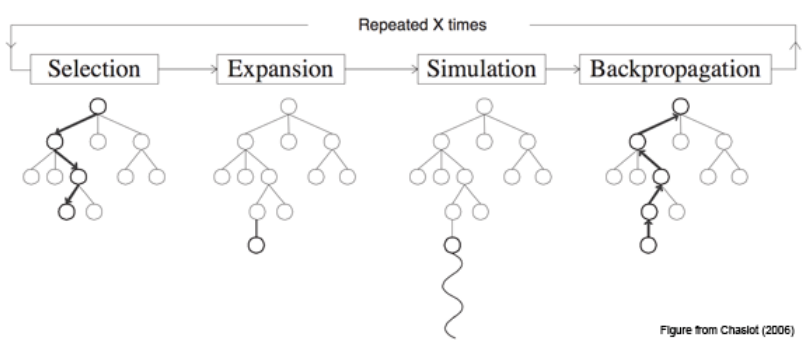
Naïve minimax vs alpha-beta pruning: AB pruning eliminates branches of the search tree where a possibility has been found which is worse than a previously examined move, meaning this branch cannot possibly influence the final decision. These traditional methods work well for e.g. chess, checkers, but are generally insufficient for more complex games that cannot be 'solved' e.g. imperfect information games such as poker.

Checkers state complexity: 10^{20} (relatively low complexity, weakly solved with traditional algorithms)
Chess state complexity: 10^{47} (higher complexity, partially solved e.g. with retrograde algorithms. May be impossible to solve with current technology)
Go state complexity: 10^{171} (very high complexity, unlikely for strong AI to come out for many years)
1940s: Monte Carlo methods formalised. 2006: Remi Coulom proposed Monte Carlo Tree Search – run random simulations and build search trees from the results.

MCTS has quickly gained traction as a strong general purpose algorithm for AI in games due to its effective results with (if properly implemented) low space and time complexity. MCTS concentrates on analysing the most promising moves, basing the expansion of the search tree on random sampling of the search space.

The game tree in MCTS grows asymmetrically, concentrating on searching more promising branches. Because of this it achieves better results than classical algorithms in games with a high branching factor. One of the most enticing benefits of MCTS is that it requires no strategic or tactical knowledge about a problem domain other than end conditions and legal moves, making MCTS implementations flexible and applicable to a variety of problem domains with little modification.

"The basic MCTS algorithm is simplicity itself: a search tree is built, node by node, according to the outcomes of simulated playouts. The process can be broken down into the following steps:



- http://www.cameronius.com/research/mcts/about/index.html "

Basic algorithm can be weak, but there are many enhancements e.g. Upper Confidence Bounds for Trees (UCT), used in 90% of MCTS applications. UCB formula:

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

"where vi is the estimated value of the node, ni is the number of the times the node has been visited and N is the total number of times that its parent has been visited. C is a tunable bias parameter. Exploitation vs Exploration The UCB formula provides a good balance between the exploitation of known rewards and the exploration of relatively unvisited nodes to encourage their exercise. Reward estimates are based on random simulations, so nodes must be visited a number of times before these estimates become reliable; MCTS estimates will typically be unreliable at the start of a search but converge to reliable estimates given sufficient time and perfect estimates given infinite time."

Improvements to MCTS? Light playouts – random moves. Heavy playouts utilise heuristics to influence choice of moves. "MCTS and UCT Kocsis and Szepervari (2006) first formalised a complete MCTS algorithm using UCB and dubbed it the Upper Confidence Bounds for Trees (UCT) method. This is the algorithm used in the vast majority of current MCTS implementations. UCT may be described as a special case of MCTS, that is: UCT = MCTS + UCB"

"Previous work has adapted MCTS to games which, like Spades, involve hidden information. This has led to the development of the Information Set Monte Carlo Tree Search (ISMCTS) family of algorithms (Cowling, Powley, and Whitehouse 2012). ISMCTS achieves a higher win rate than a knowledge-based AI developed by AI Factory for the Chinese card game Dou Di Zhu, and also performs well in other domains. ISMCTS uses determinizations, randomisations of the current game state which correspond to guessing hidden information. Each determinization is a game state that could conceivably be the actual current state, given the AI player's observations so far."

- http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewFile/7369/7595

ISMTCS useful for games like traditional Texas Hold'Em poker where each player is privy to information that others are not – i.e. their own cards. ISMTCS can model various possible game states/ permutations of what other players could have – guessing other players cards based on previous information. Not needed for OFCP because all players have the same information – cards are placed face up. However, could come into play for custom variants of OFC, such as pineapple – e.g. to model which cards are unlikely to have appeared based on how they play; if a player has a king on bottom row but does not pair it in the next hand then it is almost certain that the discarded card is not a King. Information can be built in this way to influence determined probabilities of certain cards appearing and AI can act appropriately.

2.2 Artificial Intelligence in Poker

Pot limit poker solved. Not true for holdem. Where does OFC stand? Relatively low complexity for standard OFC, but other variants of OFC are much more complex e.g. Pineapple OFC.

That said, permutations for OFCP game state: deck of 52 cards, each player places 13 cards (26 total for a heads up game)

52 choose 26 = 495,918,532,948,104 (4.9591853e+14)

Methods e,g, database look ups impractical to implement due to space complexity of game. Need a method which has a suitable compromise between time and space complexity. Monte CarloTree Search is perfect for this, especially considering the usage of heuristics which can optimise the algorithm e.g. UCT, pruning tree branches

http://scrambledeggsontoast.github.io/2014/06/26/artificial-intelligence-ofcp/ - Haskell AI for OFC 'Kachushi'. Carries out monte carlo simulations of rest of game to inform expected value for decisions.

2.3 Hand Evaluation Algorithms

Naïve – non-optimal and non-trivial to implement. Simple histogram algorithm can be used to rank high card, pair, two pair, trips, full house, quads, but needs extra steps to check flushes, straights, kickers etc. -> use this approach for bespoke 3 card evaluator for top row in OFC. Very low complexity, efficiency not as much a concern as with 5 card hands.

A faster, more efficient algorithm means more hands can be evaluated more quickly leading to higher responsiveness and more optimal play – e.g. able to evaluate more hands in deeper, broader searches. Using "kmanley"'s 5 card poker hand evaluator (handles all hands properly e.g with kickers etc, reasonable efficiency compared to other algorithms, understandable, performs better than other simple naive algorithms). Written in python so can be easily used in conjunction with my backend (cherrypy server)

3 Design and Implementation

3.1 Approach

html css and javascript for website/ app. Javascript handles front end, sends POST 'reqwest's to cherrypy server – backend – which calls helper functions (python scripts) for hand evaluation, handling AI simulations etc.

3.2 Requirements

1. OFCP application must implement appropriate rules. E.g. correct scoring system
2. AI must have a level of sophistication such that it performs well vs humans and other AI
3. AI algorithms must be suitably optimised
4. Website should have minimal downtime
5. Footprint of application must be low – if it were to be scaled up e.g. hundreds of concurrent users, server has to be able to handle this

3.3 Technologies

Website and application created with HTML, CSS and Javascript. [SQL Database for moves/ history?], python backend + cherrypy networking Hosted using VPS running Ubuntu 14.04 with nginx Version control with github: OFCP-AI private repository

3.4 Implementation

http://www.alastairkerr.co.uk/OFCP_game.html

4 Evaluation

4.1 Unit Tests

Modular testing of code e.g. individual functions

```
if (type(hand) is not str):
    print "Invalid hand (required type = string), " + str(hand) + " is " + str(type(hand)) + "\n"
    return None

if (len(hand) != 15):
    print "Invalid Hand. Required format e.g: c09c10c11c12c13 (clubs straight flush 9->King).\n"
    return None

try:
    print ('Reading in hand: ' + str(hand) + '. Reformatting now...\n' )
    cards_list = []
    formatted_hand = ""

    rank_dic = {'10':'T', '11':'J', '12':'Q', '13':'K', '14':'A'}

    for i in xrange(0,15,3):            # decode string to get each card name. index 0 -> 14 step 3
        suit = hand[i]
        rank_p1 = hand[i+1]
        rank_p2 = hand[i+2]

        suit = suit.upper()             # evaluator needs suit as uppercase char

        if suit not in ('H','D','S','C'):
            print "Invalid suit! Expected H, D, S or C. Actual:", suit
            return None

        rank = int(rank_p1 + rank_p2)   # get numerical value for rank
        if ( rank < 1 or rank > 14):
            print "Invalid rank. Accepted range 1-14.\n"
            return None
```

Fig 4.1.1 - Sample code from function 'reformat_hand_xyy_yx' in helpers.py script: use of input validation and try except blocks to catch errors

```
test_items = ( 'c05c06c07c08c09','s05c05h09s08d13','h13c01s03d05c07','invalid',100,'fakestring',('i','am','invalid'),'123456789112345' )
for item in test_items:
    format_resp = helpers.reformat_hand_xyy_yx(item)
    if format_resp != None:
        print 'Formatted ' + str(item) + ' -> ' + str(format_resp) + '\n'
```

Fig 4.1.2 - Test inputs to ensure function works as intended

```
Reading in hand: c05c06c07c08c09. Reformatting now...

[['5', 'C'], ['6', 'C'], ['7', 'C'], ['8', 'C'], ['9', 'C']]
[['5', 'C'], ['6', 'C'], ['7', 'C'], ['8', 'C'], ['9', 'C']]
Formatted c05c06c07c08c09 -> 5C6C7C8C9C

Reading in hand: s05c05h09s08d13. Reformatting now...

[['5', 'S'], ['5', 'C'], ['9', 'H'], ['8', 'S'], ['13', 'D']]
[['5', 'S'], ['5', 'C'], ['8', 'S'], ['9', 'H'], ['13', 'D']]
Formatted s05c05h09s08d13 -> 5S5C8S9HKD

Reading in hand: h13c01s03d05c07. Reformatting now...

[['13', 'H'], ['14', 'C'], ['3', 'S'], ['5', 'D'], ['7', 'C']]
[['3', 'S'], ['5', 'D'], ['7', 'C'], ['13', 'H'], ['14', 'C']]
Formatted h13c01s03d05c07 -> 3S5D7CKHAC

Invalid Hand. Required format e.g: c09c10c11c12c13 (clubs straight flush 9->King).

Invalid hand (required type = string), 100 is <type 'int'>

Invalid Hand. Required format e.g: c09c10c11c12c13 (clubs straight flush 9->King).

Invalid hand (required type = string), ('i', 'am', 'invalid') is <type 'tuple'>

Reading in hand: 123456789112345. Reformatting now...

Invalid suit! Expected H, D, S or C. Actual: 1
127.0.0.1 - - [04/Apr/2015:22:23:57] "POST /subpage/eval-one-hand-test/ HTTP/1.0" 200 766 "
http://alastairkerr.co.uk/OFCP_game.html" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gec
ko/20100101 Firefox/36.0"
```

Fig 4.1.3 - Output – invalid hands are handled properly, throw exceptions/ print usage messages rather than throwing errors

4.2 Performance of AI versus human players

Alpha testing: playing individual games with participants vs AI Playing vs experienced players, new players – get an indication of AI's comparative skill level

4.3 Performance of AI versus other AI

Pit this intelligent agent vs other AI and/or previous/ alternative versions of itself. E.g. performance of AI with MCTS vs AI using AB pruning/ minimax, MCTS vs totally random placement: if AI is working well should vastly outperform a naive AI. Visualisations of performance e.g. graphs, tables of win rates Database storing moves -> this would allow for analysis of individual rounds, games etc.

5 Conclusion

5.1 Aims and Objectives

To what extent were the aims met? Sophistication and performance of AI? Were all features implemented?

5.2 Reflection

Reflection on project, decisions, performance etc.

5.3 Improvements

What can be done to improve the application/ AI in the future?

Limitations: “With any method based on random simulation, it is inevitable that poor quality moves will be chosen with nonzero probability, due to a particularly lucky run of simulations making the move appear better than it is. “ - http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewFile/7369/7595 page 5

Improve frontend – make the app more visually appealing

Multiplayer support for players vs players as well as players vs AI(s) or players vs players vs AI(s)

Add support for other variants of OFC such as pineapple

References

[1] Browne, C. Et al, 2012, A Survey of Monte Carlo Tree Search Methods, IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 4, NO. 1.

[2] Cowling, P. Et al, 2012, Information Set Monte Carlo Tree Search, IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 4, NO. 2.

[3] Whitehouse, D. Et al, 2013, Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game, University of York

[4] Özcan, E. and Hulagu, B., 2004, A Simple Intelligent Agent for Playing Abalone Game: ABLA, Proc. of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks, pp. 281-290

[5] http://scrambledeggsontoast.github.io/2014/06/26/artificial-intelligence-ofcp/ - Intelligent Agent ‘Kachushi’ for Open Face Chinese Poker using Monte Carlo methods, implemented in Haskell, accessed 2015